Local Users Guide to Theorist*

David Poole
Department of Computer Science,
University of British Columbia,
Vancouver, B. C., Canada V6T 1W5.
(604) 228-6254
poole@cs.ubc.ca

December 15, 2009

Abstract

Theorist [Poole88, PGA87, Poole89, Poole90] is a logical reasoning system for default reasoning and diagnosis. It is based on the idea of theory formation from a fixed set of possible hypotheses. This is a users manual for a compiler from Theorist to Prolog. This code is as described in [Poole91].

This compiler should not be seen as "the" theorist system, but of one (rather simple and naive) implementation of the Theoirst framework. For example, it finds all explanations of the given goal, and does no simplification of the explanations.

Contents

1 Introduction **2**

2 Theorist Framework **2**

2.1 Syntax 3

2.2 Compiler Directives 5

2.3 Flags 5

*Copyright ©1990 David Poole. All rights reserved.

3 Examples **6**
3.1 Default Reasoning 6
3.2 Cancelling Defaults 8
3.3 Disjunctive Answers 9
3.4 circuits 10
3.5 Prediction 13

4 Conclusion **15**

1 Introduction

This paper describes the use of the Theorist compiler.

To run this you must have access to a machine with Sicstus Prolog or Quintus Prolog, or many other Prologs that are similar. It is meant to be portable.

When you execute this, you can type Theorist commands at it. N.B. you can also type Prolog queries at it; the Theorist commands will just be Prolog queries.

The source code is in the file

```
theorist.tex
```

This can either be consulted as a Prolog program or typeset as a Latex document with only one character editing at the start of the file.

2 Theorist Framework

Theorist [Poole88, PGA87, Poole89, Poole90] is a logical reasoning system for default reasoning and diagnosis. It is based on the idea of theory formation from a fixed set of possible hypotheses.

This implementation is of the version of Theorist described in [Poole91]. The user provides three sets of first order formulae

$\mathcal{F}$ is a set of closed formulae called the *facts*. These are intended to be true in the world being modelled.

Δ is a set of formulae which act as *possible hypotheses*, any ground instance of which can be used in an explanation if consistent.

$\mathcal{C}$ is a set of closed formulae taken as constraints. The constraints restrict what can be hypothesised.

We assume that $\mathcal{F} \cup C$ is consistent.

Definition 1 a **scenario** of $\mathcal{F}, \Delta$ is a set $D \cup \mathcal{F}$ where D is a set of ground instances of elements of Δ such that $D \cup \mathcal{F} \cup C$ is consistent.

Definition 2 If g is a closed formula then an **explanation** of g from $\mathcal{F}, \Delta$ is a scenario of $\mathcal{F}, \Delta$ which implies g.

That is, g is explainable from $\mathcal{F}, \Delta$ if there is a set D of ground instances of elements of Δ such that

$\mathcal{F} \cup D \models g$ and
$\mathcal{F} \cup D \cup C$ is consistent

$\mathcal{F} \cup D$ is an explanation of g.

In other papers we have described how this can be the basis of default and abductive reasoning systems [PGA87, Poole88, Poole89, Poole90]. If we are using this for prediction then possible hypotheses can be seen as defaults. [Poole88] describes how this formalism can account for default reasoning. This is also a framework for abductive reasoning where the possible hypotheses are the base causes we are prepared to accept as to why some observation was made [PGA87, Poole89, Poole90]. We will refer to possible hypotheses as defaults.

One restriction that can be made with no loss of expressive power is to restrict possible hypotheses to just atomic forms with no structure [Poole88]. This is done by naming the defaults.

2.1 Syntax

The syntax of Theorist is designed to be of maximum flexibility. Virtually any syntax is appropriate for wffs; the formulae are translated into Prolog clauses without mapping out subterms. The theorem prover implemented in the Compiler can be seen as a non-clausal theorem prover [Poole84].

A **wff** is a well formed formula made up of arbitrary combination of implication ("=>", "< −"), disjunction ("*or*", ";"), conjunction ("*and*",

"&", ",") and negation ("*not*", "~") of atomic symbols. Variables follow the Prolog convention of being in upper case. There is no explicit quantification.

A **name** is an atomic symbol with only free variables as arguments.

The following gives the syntax of the Theorist code:

fact w.
: where w is a wff, means that $(\forall w) \in \mathcal{F}$; i.e., the universal closure of w (all variables universally quantified) is a fact.

default d.
: where d is a name, means that $d \in \Delta$; i.e., d is a default (a possible hypothesis).

default $d : w$.
: where d is a name and w is a wff means w, with name d can be used in a scenario if it is consistent. Formally it means $d \in \Delta$ and $(\forall d \Rightarrow w) \in \mathcal{F}$.

constraint w.
: where w is a wff means $\forall w \in \mathcal{C}$.

prolog p.
: where p is an atomic symbol means any Theorist call to p should be proven in Prolog. This allows us to use built-in predicates of pure Prolog. One should not expect Prolog's control predicates to work.

explain w.
: where w is an arbitrary wff, gives all explanations of $\exists w$.

predict w.
: where w is a arbitrary ground wff, returns "yes" if w is in every extension of the defaults and "no" otherwise. If it returns "yes", a set of explanations is returned, if it returns "no" then a scenario from which g cannot be explained is returned (this follows the framework of [Poole89]).

In this compiler these are interpreted as commands to Prolog. The interface will thus be the Prolog interface with some predefined commands.

2.2 Compiler Directives

The following are compiler directives:

thconsult *filename.*
reads commands from *filename*, and asserts and/or executes them.

thtrans *filename.*
reads commands from *filename* and translates them into Prolog code in the file *filename.pl*.

thcompile *filename.*
reads commands from *filename*, translates them into the file *filename.pl* and then compiles this file. "explain" commands in the file are not interpreted.

dyn *atom.*
should be in a file and declares that anything matching the atom is allowed to be asked or added to. This should appear before any use of the atom. This corresponds to the "dynamic" declaration of Quintus Prolog. This is ignored except when compiling a file.

2.3 Flags

There are some other commands which allow one to set flags. There are a number of Theorist options which can be set by flag declarations. Flags, by default, are `on`. To set the flag f to value v you can issue the command

set f, v.

To find out the value of the flag f issue the command

flag f, V.

You can reset the value of flag f to its old value by

reset f.

The following flags currently do something:

ancestor_search when on means that we check for ancestors. If this is off we lose completeness for the general formula case. If we are only using Horn clauses then we do not need an ancestor search.

loop_check this turns on a simple loop check. It makes the system propositionally complete. The overhead for long loops may be very high though. If your program runs very slowly, then turn it off. Often dramatic improvements can be acheived in this way. It does not affect correctness of answers found.

asserting when on, this asserts the compiled clauses into Prolog, when off it writes the Prolog form of the assertions onto the standard output. This is mainly used internally, but is useful for seeing the output of the Theorist compiler.

3 Examples

This section contain many examples. The best source of descriptions of more examples is [Poole90].

3.1 Default Reasoning

The following example is a file to do the birdsfly example

```
dyn flies(X).
dyn bird(X).
default birdsfly(X): flies(X) <- bird(X).
constraint not birdsfly(X) <- not flies(X).
default emusdontfly(X): not flies(X) <- emu(X) .
constraint not emusdontfly(X) <- flies(X).
constraint not birdsfly(X) <- emu(X).
fact bird(X) <- emu(X).
fact bird(X) <- robin(X).
fact bird(tweety).
fact emu(polly).
fact robin(cohen).
```

The following is a session with theorist. We assume that the preceeding code is in a file called "birdsfly".

```
 theorist

Welcome to THEORIST 1.1 (5 October 88 version)
Any Problems see David Poole (poole@vision.ubc.cdn)

yes
| ?- thconsult birdsfly.

yes
| ?- explain flies(tweety).
Answer is flies(tweety)
Theory is [birdsfly(tweety)]


no
| ?- explain flies(polly).

no
| ?- explain flies(X).
Answer is flies(cohen)
Theory is [birdsfly(cohen)]

Answer is flies(tweety)
Theory is [birdsfly(tweety)]


no
| ?-
```

Note that the explain command always finds all answers and then returns with "no" (note that, for simplicity, it is a Prolog command). Note also that we did a "thconsult"; we could have also done a "thcompile" but it would have taken longer to load the file, but would have executed quicker. Note also that if we had typed the commands at the terminal (which is the same

a "thconsult"ing them, but not "thcompile"ing them, then we did not need the "dyn" declaration).

3.2 Cancelling Defaults

We can do a more complicated example to show cancelling defaults

```
dyn flies(X).
default mammals_dont_fly(X): not flies(X) <- mammal(X).
constraint not mammals_dont_fly(X) <- flies(X).
default bats_fly(X): flies(X) <- bat(X).
constraint not bats_fly(X) <- not flies(X).
constraint not mammals_dont_fly(X) <- bat(X).
default dead_things_dont_fly(X): not flies(X) <- dead(X).
constraint not dead_things_dont_fly(X) <- flies(X).
constraint not bats_fly(X) <- dead(X).
fact mammal(X) <- bat(X).
fact mammal(bruce).
fact bat(paul).
fact bat(dracula).
fact dead(dracula).
```

Give that file we can do the following

```
| ?- explain flies(X).
Answer is flies(paul)
Theory is [bats_fly(paul)]

no
| ?- explain not flies(X).
Answer is not flies(bruce)
Theory is [mammals_dont_fly(bruce)]

Answer is not flies(dracula)
Theory is [dead_things_dont_fly(dracula)]

no
```

3.3 Disjunctive Answers

Consider the following session

```
| ?- fact p(a) or p(b).

yes
| ?- fact p(X) and not q(f(X)) => q(X).

X = _38

| ?- explain p(X).

Answer is p(a) or p(b)
Theory is []

Answer is p(b) or p(a)
Theory is []

no
| ?- explain q(X).

Answer is q(a) or q(f(b)) or q(b) or q(f(a))
Theory is []

Answer is q(b) or q(f(a)) or q(a) or q(f(b))
Theory is []

Answer is q(f(a)) or q(f(b)) or q(b) or q(a)
Theory is []

Answer is q(f(b)) or q(f(a)) or q(a) or q(b)
Theory is []

no
```

Note that we get different orderings from the disjunct. This is because the linear resolution method used finds each of the proofs with each of the

disjuncts as head, and we do not simplify the results.

3.4 circuits

This specifies an abductive program for the full adder:

```
% a circuit diagnosis program for a full adder
dyn diag(X,Y).
dyn val(X,Y).
% to use add in facts about the input of the circuit, such as
% fact val(in(1, f1), off).
% fact val(in(2, f1), off).
% fact val(in(3, f1), off).
% to get it to explain the output of the circuit do something lik
% diag(on, on).

fact gate(x1, xor).
fact gate(x2, xor).
fact gate(a1, and).
fact gate(a2, and).
fact gate(o1, or).

fact conn(in(1, f1), in(1, x1)).
fact conn(in(1, f1), in(1, a1)).
fact conn(in(2, f1), in(2, x1)).
fact conn(in(2, f1), in(2, a1)).
fact conn(in(3, f1), in(2, x2)).
fact conn(in(3, f1), in(1, a2)).
fact conn(out(1, x1), in(1, x2)).
fact conn(out(1, x1), in(2, a2)).
fact conn(out(1, a1), in(2, o1)).
fact conn(out(1, a2), in(1, o1)).
fact conn(out(1, x2), out(1, f1)).
fact conn(out(1, o1), out(2, f1)).

fact val(in(N, Device), anything).
```

```
default ok(Device).

fact val(out(1, Device), Out1) <-
ok(Device),
gate(Device, Type),
ttable(Type, In1, In2, Out1),
val(in(1, Device), In1),
val(in(2, Device), In2).

default faulty(Device).

fact val(out(1, Device), Out1) <-
faulty(Device),
gate(Device, Type),
ttable(Type, In1, In2, Eout),
opp(Eout, Out1),
val(in(1, Device), In1),
val(in(2, Device), In2).

fact n(ok(Device)) <- faulty(Device).

fact opp(on, off).
fact opp(off, on).

fact ttable(and, on, on, on).
fact ttable(and, off, anything, off).
fact ttable(and, anything, off, off).
fact ttable(or, off, off, off).
fact ttable(or, on, anything, on).
fact ttable(or, anything, on, on).
fact ttable(xor, off, on, on).
fact ttable(xor, off, off, off).
fact ttable(xor, on, X, Y) <- opp(X, Y).

fact val(Y, Z) <-
ne(Z, anything),
conn(X, Y),
```

```
val(X, Z).

% define enough of "ne" - not equals - to make this work
ne(on,anything ).
ne(off,anything ).

fact n(val(X, off)) <- val(X, on).

fact diag(Out1, Out2) <-
val(out(1, f1), Out1 ),
val(out(2, f1), Out2 ).
```

We can do circuit diagnosis:

```
| ?- thcompile cir2.
[compiling /ubc-lcvfs1/poole/theorist/cir2.pl...]
[cir2.pl compiled 10.133 sec 29,868 bytes]

yes
| ?- fact val(in(1, f1), off).

yes
| ?- fact val(in(2, f1), off).

yes
| ?- fact val(in(3, f1), off).

yes
| ?- explain diag(on,on).

Answer is diag(on,on)
Theory is [faulty(a2),ok(o1),faulty(x1),ok(x2)]

Answer is diag(on,on)
Theory is [faulty(a1),ok(o1),faulty(x1),ok(x2)]

Answer is diag(on,on)
```

```
Theory is [faulty(a1),ok(o1),faulty(x1),ok(x2)]

Answer is diag(on,on)
Theory is [ok(a1),ok(a2),faulty(o1),faulty(x1),ok(x2)]

Answer is diag(on,on)
Theory is [ok(a1),ok(a2),faulty(o1),faulty(x1),ok(x2)]

Answer is diag(on,on)
Theory is [faulty(a2),ok(o1),ok(x1),faulty(x2)]

Answer is diag(on,on)
Theory is [faulty(a2),ok(o1),ok(x1),faulty(x2)]

Answer is diag(on,on)
Theory is [faulty(a1),ok(o1),ok(x1),faulty(x2)]

Answer is diag(on,on)
Theory is [faulty(a1),ok(o1),ok(x1),faulty(x2)]

Answer is diag(on,on)
Theory is [ok(a1),ok(a2),faulty(o1),ok(x1),faulty(x2)]

Answer is diag(on,on)
Theory is [ok(a1),ok(a2),faulty(o1),ok(x1),faulty(x2)]

Answer is diag(on,on)
Theory is [ok(a1),ok(a2),faulty(o1),ok(x1),faulty(x2)]

Answer is diag(on,on)
Theory is [ok(a1),ok(a2),faulty(o1),ok(x1),faulty(x2)]

no
```

3.5 Prediction

We predict what is in every extension.

```
default quaker_so_dove(X): quaker(X) => dove(X).
default republican_so_hawk(X): republican(X) => hawk(X).
default hawk_so_motivated(X): hawk(X) => motivated(X).
default dove_so_motivated(X): dove(X) => motivated(X).
default dove_so_peaceful(X): dove(X) => peaceful(X).
fact not (hawk(X) and dove(X)).
fact republican(ron).
fact quaker(george).
fact republican(dick).
fact quaker(dick).
```

The following tracing shows how prediction works.

```
| ?- predict motivated(ron).
Yes, motivated(ron) is in all extensions.
Explanations are:
1: [hawk_so_motivated(ron),republican_so_hawk(ron)].

yes
| ?- predict hawk(ron).
Yes, hawk(ron) is in all extensions.
Explanations are:
1: [republican_so_hawk(ron)].

yes
| ?- predict motivated(dick).
Yes, motivated(dick) is in all extensions.
Explanations are:
1: [dove_so_motivated(dick),quaker_so_dove(dick)].
2: [hawk_so_motivated(dick),republican_so_hawk(dick)].

yes
| ?- predict hawk(dick).
No, hawk(dick) is not explainable from [quaker_so_dove(dick)].

yes
| ?- predict peaceful(dick).
No, peaceful(dick) is not explainable from [republican_so_hawk(dick)].
```

```
yes
| ?- predict peaceful(george).
Yes, peaceful(george) is in all extensions.
Explanations are:
1: [dove_so_peaceful(george),quaker_so_dove(george)].

yes
```

4 Conclusion

Any comments about the product or the manual please direct to the author. Have fun.

Acknowledgements

This work could not have been done without the ideas, criticism and feedback from Randy Goebel, Eric Neufeld, Paul Van Arragon, Scott Goodwin and Denis Gagné. Thanks to Brenda Parsons and Amar Shan for valuable comments on a previous version of this paper. This research was supported under NSERC grant A6260.

References

[Poole84] D. Poole, "Making Clausal theorem provers Non-clausal", *Proc. CSCSI-84*. pp. 124-125.

[Poole88] D. Poole, "A Logical Framework for Default Reasoning", *Artificial Intelligence* 36(1), 27–47, 1988.

[Poole89] D. Poole, "Explanation and prediction: an architecture for default and abductive reasoning", *Computational Intelligence* 5(2), 97–110, 1989.

[Poole90] D. Poole, "A methodology for using a default and abductive reasoning system", *International Journal of Intelligent Systems* 5(5), 521–548, 1990.

[Poole91] D. Poole, "Compiling a Default Reasoning System into Prolog", *New Generation Computing*, 9(1), 3–38, 1991.

[PGA87] D. L. Poole, R. G. Goebel and R. Aleliunas, "Theorist: A Logical Reasoning System for Defaults and Diagnosis", in N. Cercone and G. McCalla (Eds.) *The Knowledge Frontier: Essays in the Representation of Knowledge*, Springer Varlag, New York, 1987, pp. 331-352.